9.1 INTRODUCTION

When multiple transactions are running at the same time, then it may happen that various transactions may interfere with each other. For example, multiple transactions updating a data object at the same time. This may eventually lead to data becoming inconsistent. So, a mechanism is required to ensure that these transactions do not interfere with each other. Concurrency control is a technique that attempts to interleave various operations of multiple transactions so that the results are same as the results of a serial schedule execution.

9.2 CONCURRENCY PROBLEMS

(GGSIPU 2010)

Problems of concurrency are classified into:

(a) **Lost update problem:** Suppose there are two transactions T1 and T2. Both these transactions want to read the data object and then update it. Both transactions first read the same data at time t_1 and t_2 respectively and then update it at time t_3 and t_4 respectively. Now update done by transaction T1 will be overwritten by transaction T_2 which implies update done by T1 is lost.

	T_1	T_2
t_1	read (A)	
t_2		read(A)
t_3	write (A)	
t_4		write (A)

Fig. 9.1: Lost Update Problem

(b) **Dirty read (Uncommitted data) problem:**

	T_1	T_2	Database
t_1		A = A + 2 write (A)	A = 10 A = 12
t_2	read(A) A = A + 5 write (A)		A = 12 A = 17
t_3		roll back	A = 10

Fig. 9.2: Dirty Read Problem

In the above schedule, transaction T1 at time t_2 reads the updated value of A and further changes it to 17. Now, at time t_3 transaction T2 rollbacks so, the update done by T2 (A=12) at time t_1 disappears. This means T1 has read and updated a faulty value. This reading of faulty value is called dirty read.

(c) **Unrepeatable read (Inconsistent retrievals) problems:**

	T_1	T_2	Database
			A = 10 B = 10 C = 10 sum = 0
t_1	read (A) sum = sum + A		sum = 10
t_2	read (B) sum = sum + B		sum = 20
t_3		read (A)	A = 10
t_4		A = A + 20	
t_5		Commit	A = 30 B = 10 C = 10 sum = 20
t_6	read (C) sum = sum + C		sum = 30

Fig. 9.3: Unrepeatable read problem

Suppose transaction T1 want to calculate the sum of value of A, B and C. It performs three reads and three add operations at time t_1, t_2 and t_6. In the meantime, transaction T2 reads and updates value of A. Transaction T1 calculate the sum as 30 but does not realise that value of A has been changed and actual sum should be 50.

Incorrect Summary problem

9.3 LOCK BASED PROTOCOL

(GGSIPU, 2011; MDU, May 2011, PTU 2010)

Locking is a technique that is used for concurrency control. Whenever a transaction wants to perform any operation on a data object, it needs to acquire a lock on that data object. Acquiring lock on that data object means that when one transaction is updating the data then no other transaction is allowed to access that same data.

Various types of locking techniques are:

(a) Binary Locking

(GGSIPU, 2013)

There are two states of binary lock namely locked ('1') or unlocked ('0'). Each and every data object in the database is associated with a lock. Whenever a transaction wants to access a data object (suppose A), it firstly have to request for a lock on object A. If lock (A) = 1, then transaction has to wait because some other

transaction is currently performing an operation on A. But, if lock (A) = 0, then transaction can access object A by issuing lock_item (A) request. This request when granted makes lock (A) = 1. After transaction has finished accessing the object, it has to issue an unlock_item (A) request which make lock (A) = 0. Object A can now be accessed by other transaction.

(b) Shared/Exclusive Locking

Shared Lock(s). If a transaction T_i want to just read a data object A, then it should request for a shared mode lock. If this lock is granted then T_i will be able to only read A but cannot write A.

Exclusive Lock (x). If a transaction T_i wants to perform both read and write operation on a data object A, then it should request for a exclusive mode lock on A.

Rules for Shared/Exclusive Lock Mode

1. If a transaction T_i locks a data object A in shared mode, then any other transaction T_j can acquire an shared lock on A, but it cannot acquire an exclusive lock on A.
2. If a transaction T_i locks a data object A in exclusive mode, then any other transaction T_j cannot acquire either an exclusive lock or shared lock on A.
3. When a transaction finishes access to an object, it should unlock that object.

Example: We will now apply this locking scheme to lost update problem.

T_1	T_2
s_lock(A) read(A)	
	s_lock(A) read(A)
request x_lock(A) wait	
unlock(A)	
	unlock(A)
x_lock(A) write(A)	
	request x_lock(A) wait
unlock (A)	
	x_lock(A) write (A) unlock(A)

Fig. 9.4: Example of Shared/Exclusive Lock Mode

(c) **Two Phase Locking Protocol.** Every transaction issues a lock and unlock request in two phases: **(GGSIPU, 2011; UPTU 2004, 2006, 2008; PTU 2012)**

(i) ***Growing Phase.*** In this phase transaction can only obtain new lock but cannot release any lock.

(ii) ***Shrinking Phase.*** A transaction release all locks obtained by it, but cannot obtain a new lock.

Initially a transaction is in growing phase when it obtains all locks. After reaching the lock point (point at which final lock is obtained) transaction enters in shrinking phase, where it has to release all locks.

Example: The following schedule obeys two phase locking.

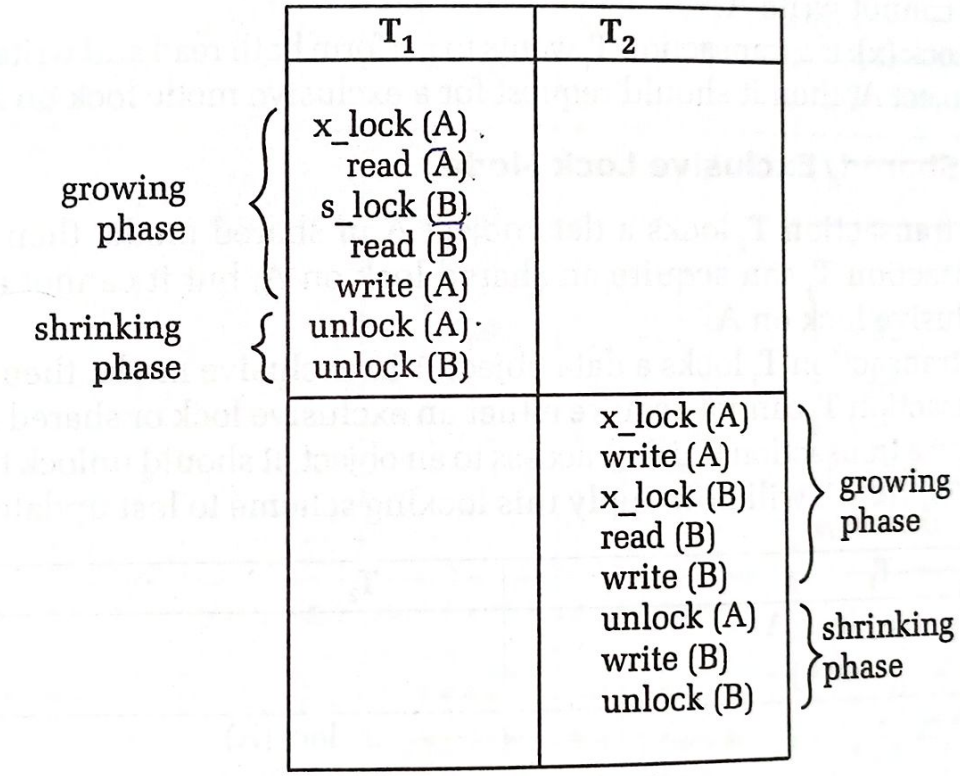

Fig. 9.5: Two Phase Locking

Advantages of Two Phase Locking

(i) Two phase locking ensures serialisability

Disadvantages of Two Phase Locking

(i) It can lead to deadlock.

(ii) It may cause cascading roll back. Suppose transaction T1 writes a value which is used by transaction T2. If T1 fails it has to be rolled back. Now, since T2 depends on T1, T2 also have to be rolled back.

Two Phase Locking Protocol has two varients:

(i) ***Strict two phase locking.*** All exclusive lock held by a transaction must be kept until the transaction commits. This avoid cascade rollback problems as data written by an uncommitted transaction are locked in exclusive mode until that transaction commits, preventing any other transaction from reading the data.

Advantage of Strict Two Phase Locking

- Since only cascadelers schedule are produced recovery is easy.

Disadvantage of Strict Two Phase Locking

- Since set of schedule obtained is a subset of those obtained from two phase locking, concurrency is reduced.

(ii) **Rigorous two phase locking.** It requires that all lock must be held until the transaction commits.

9.4 TIMESTAMP BASED PROTOCOLS

(GGSIPU 2007, 2010, 2012; UPTU 2006-07, 2010, 2011-12)

Whenever a transaction starts, a unique timestamp (or identifier) is assigned to that transaction based on the time in the system clock. Let this value be TS(T). A counter can also be used for assigning timestamp which is incremented after a new timestamp is assigned. For any transactions T_i and T_j, if T_i come first, then TS (T_i) < TS(T_j). Any conflicting operations are scheduled based on their timestamp value.

Each data item X is assigned two timestamp value.

(i) ***W-timestamp (x)*** – Largest timestamp value of any transaction that executed write (x) successfully.

(ii) ***R-timestamp (x)*** – Largest timestamp value of any transaction that executed read (x) successfully.

These timestamps are updated whenever a new read (Q) or write (Q) instruction is executed. Any conflicting read and writes operations are executed as follows:

1. ***Transaction T_i issue read (x)***

   (a) If **TS (T_i) < W-timestamp (x),** then reject read (x) and roll back T_i because T_i reads a value of x which was already overwritten by another transaction.

   Explanation: Let W-timestamp (x) = 5
   TS (T_i) = 4

   T_i wanted to read X at time 4, but before T_i could read the data, some other transaction T_j, which come after T_i changed the data at time 5. Now if T_i read X it would see changed data and not the original one which it wanted to see. So, reject the read operation and roll back Ti.

   (b) If **TS (T_i)> = W-timestamp (x),** then read (x) is executed and R-timestamp (x) is replaced by maximum of R-timestamp and TS (T_i).

   Explanation: Let W-timestamp (x) = 5
   TS (T_i) = 7
   R-timestamp = 6

T_i can read the data at time 7 because it will read the value which was changed by a transaction which came before it (at time 5). Now, new R-timestamp (x) will be 7 (the latest time at which x was read).

2. **Transaction T_i issue write (x)**

(a) **If TS (T_i) < R-timestamp (x),** then write operation is rejected and T_i is rolled back because the value of x which T_i is producing was needed previously and system assumed that value would never be produced.

Explanation: Let TS (T_i) = 5

R-timestamp (x) = 6

T_i wants to change x at time 5 but before it could some other transaction T_j which came after T_i reads the unchanged value of X. If write (x) is allowed the data will become inconsistent.

(b) If **TS (T_i) < W-timestamp (x),** then write (x) is rejected and T_i is rolled back because T_i is attempting to write absolute value of x.

Explanation: Let TS (T_i) = 5

Wtimestamp (x) = 6

T_i wanted to change value of x at time 5. But some other transaction T_j which came after T_i has already changed the value of x at time 6. So, if write (x) is allowed it will change the already changed value and not the original one which it wanted to do.

(c) **Otherwise** execute write (x) and W-timestamp (x) is replaced by TS(T_i).

Explanation: T_i wants to change value of x, which was changed or read by a transaction that came after T_i and hence lower in timestamp order. so write (x) can be executed.

9.7 DEADLOCK

(GGSIPU 2009, UPTU 2006-07; UPTU 2011-12)

Deadlock occurs when there exist a set of two or more transactions and each transaction is waiting for some data item that is locked by some other transaction.

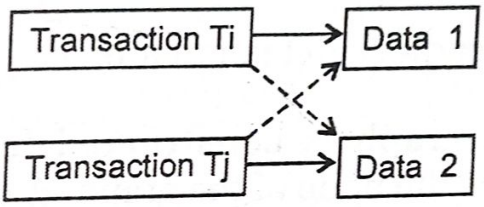

Fig. 9.10: Deadlock

Fig. shows T_i has a lock on data 1 and is waiting to acquire a lock on data 2, but lock on data 2 cannot be granted because it is already locked by T_j and T_j is waiting for T_i to unlock data 1.

The following schedule shows a deadlock.

T_1	T_2
s_lock (X) read(X)	
	s_lock (Y) read (Y)
x_lock (Y)	
	x_lock(X)

Fig. 9.11: Example of Deadlock

9.8 MULTIPLE GRANULARITY

(UPTU 2011-12)

In the locking schemes discussed so far, we did not consider the size of a data item being locked. A data item could be the entire database, a file or a record. Size or granularity of data item is considered as the major factor on the overall performance of the concurrency control scheme. If granularity of data item is very large (for e.g. an entire database), then a transaction T while accessing a few records will lock the entire database and no concurrent access by other transaction is possible even though they may refer to record not accessed by T and hence, locking overhead is low. But if the granularity of data item is small (for e.g. a field of a record), concurrency would improve.

In multiple granularity scheme small granularity are nested within larger one. Such a hierarchy is represented by a tree structure. The tree consists of four levels. The highest level represents the entire database. The children of this node are of type area. Each area in turn has nodes of type file as its children. No files are in more than one area. Finally, the lowest level represents the nodes of type records.

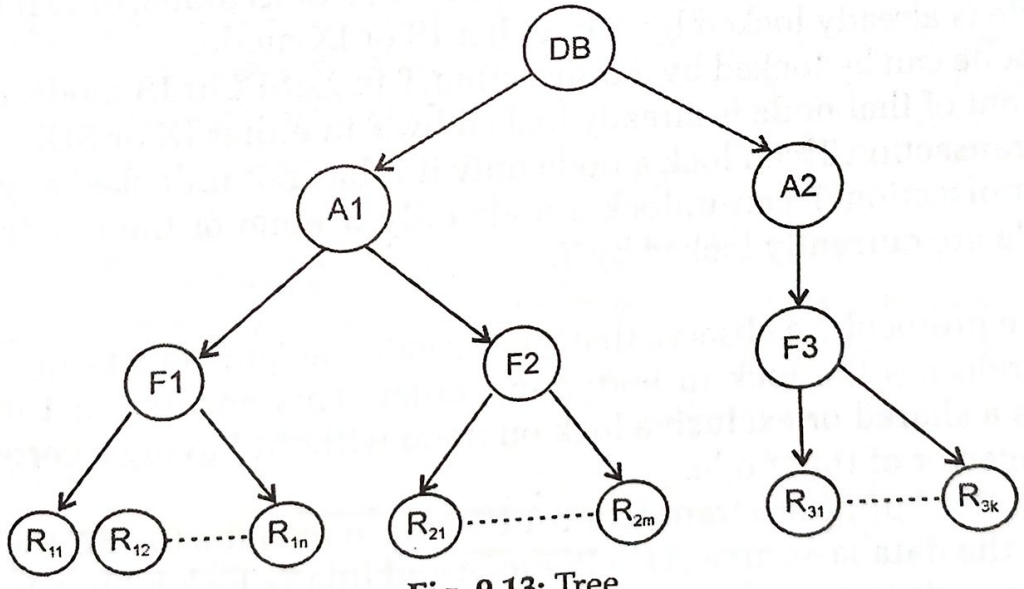

Fig. 9.13: Tree

In addition to shared and exclusive locks, multiple granularity locking protocol uses three new kinds of locks called intension shard (IS), intension exclusive (IX) and shared and intension exclusive (SIX) lock.

Moreover, it is assumed that if a transaction locks a node in any mode then, it also implicitly locks all the children of that node in the same mode.

If a node is locked in IS mode, then explicit locking is done at lower level with only shared mode locks. If a node is locked in IX mode then explicit locking is done at lower level with only exclusive or shared mode locks. If a node is locked in SIX mode, then the sub-tree rooted by that mode is locked explicitly in shared mode and that explicit locking is being done at a lower level with exclusive mode locks.

The compatibility table of the intension locks, shared lock and exclusive locks is shown by following matrix.

	S	X	IS	IX	SIX
S	Yes	No	Yes	No	No
X	No	No	No	No	No
IS	Yes	No	Yes	Yes	Yes
IX	No	No	Yes	Yes	No
SIX	No	No	Yes	No	No

Fig. 9.14: Lock Compatibility Matrix